Table of Contents

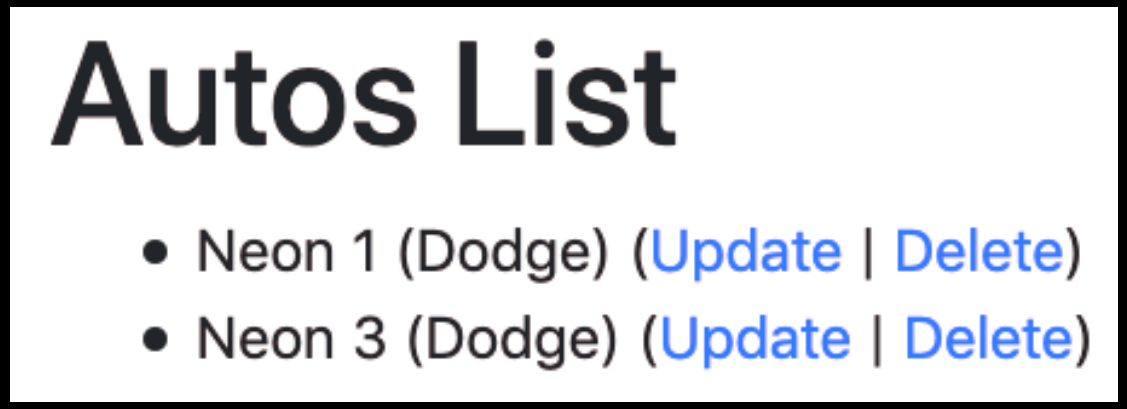
This slide deck consists of slides used in 3 lecture videos in Week 5. Below is a list of shortcut hyperlinks for you to jump into specific sections.

Charles Severance
www.dj4e.com

Django Owned Rows

https://samples.dj4e.com/myarts/
https://github.com/csev/dj4e-samples/

Who Can Edit Which Row?



In our Autos CRUD assignment, any user could edit any row. But in real systems, different users own each row in a data model and we only allow a user to edit /modify the row(s) that "belong to them".

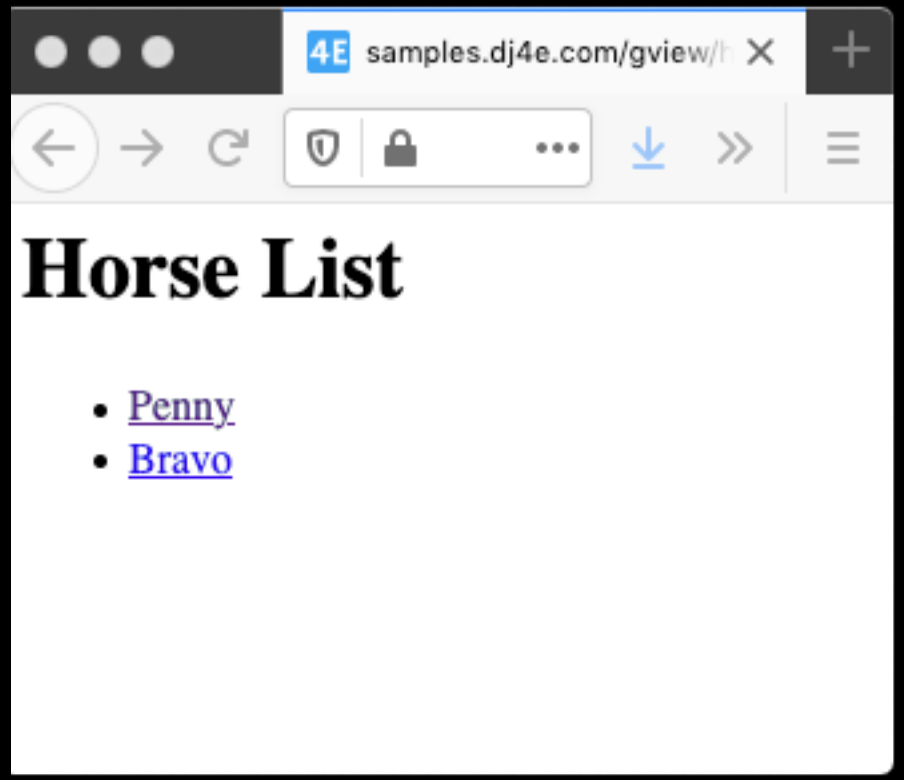
dj4e-samples/gview/views.py

```
from django.views import generic
from gview.models import Horse

class HorseListView(generic.ListView):
    model = Horse
```

dj4e-samples/gview/templates/gview/horse_list.html

```
<h1>Horse List</h1>
<p>
{% if horse_list %}
<ul>
  {% for horse in horse_list %}
    <li>
      <a href="{% url 'gview:horse' horse.id %}">
        {{ horse.name }}</a>
    </li>
  {% endfor %}
</ul>
{% else %}
  <p>There are no horses in the database.</p>
{% endif %}
</p>
```

https://samples.dj4e.com/gview/horses

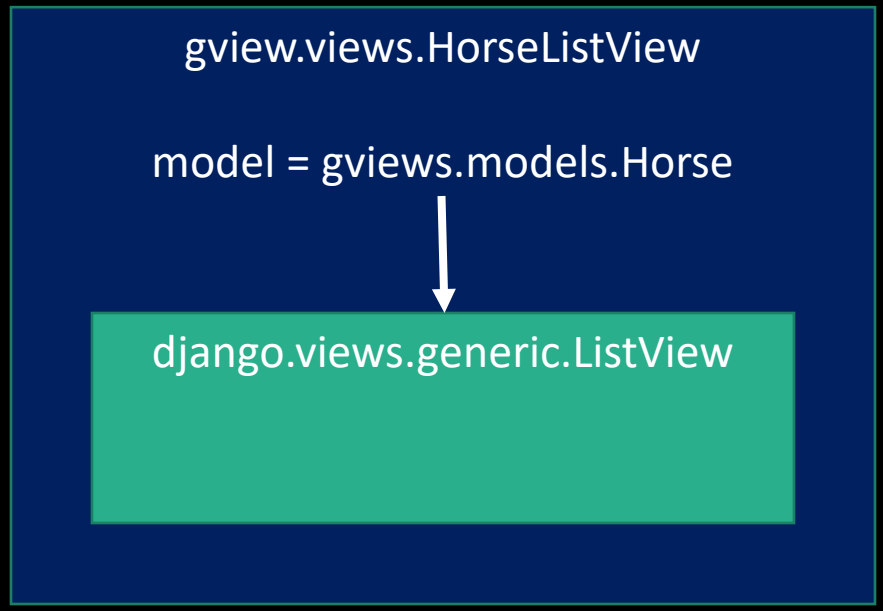


dj4e-samples/gview/views.py

```
from django.views import generic
from gview.models import Horse

class HorseListView(generic.ListView):
    model = Horse
```

dj4e-samples/gview/templates/gview/horse_list.html

```
<h1>Horse List</h1>
<p>
{% if horse_list %}
<ul>
  {% for horse in horse_list %}
    <li>
      <a href="{% url 'gview:horse' horse.id %}">
        {{ horse.name }}</a>
    </li>
  {% endfor %}
</ul>
{% else %}
  <p>There are no horses in the database.</p>
{% endif %}
</p>
```

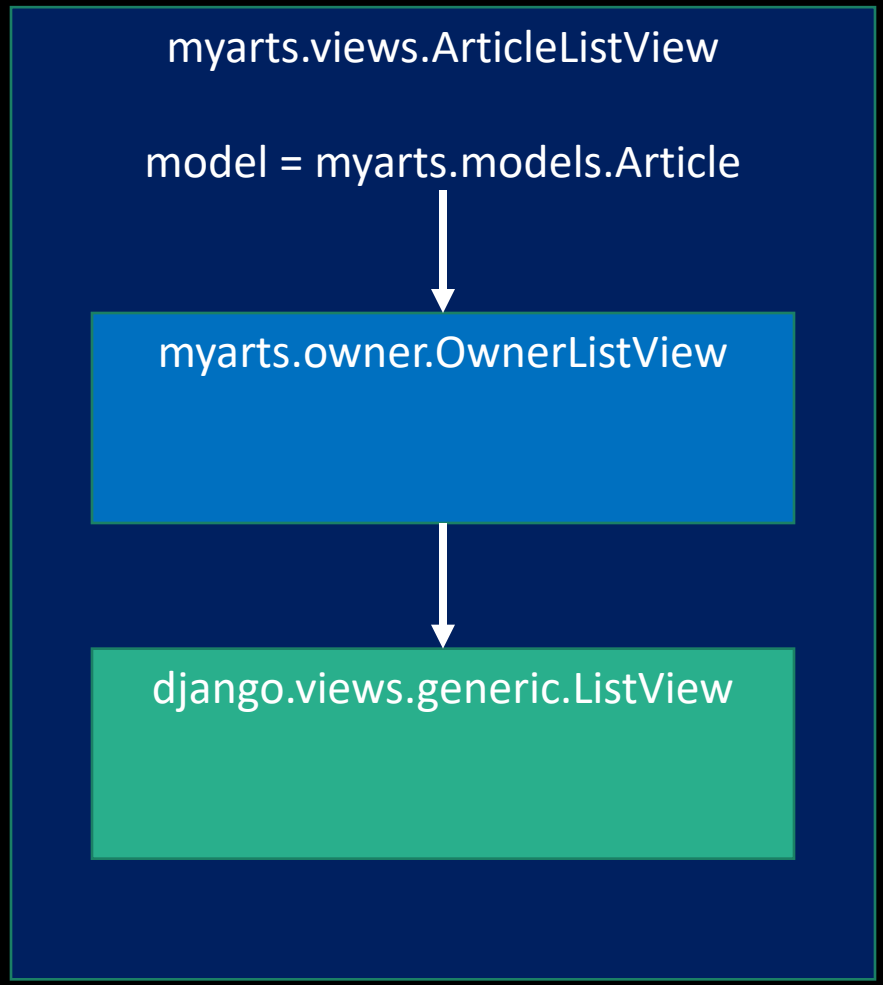


dj4e-samples/myarts/views.py

```
from myarts.models import Article
from myarts.owner import OwnerListView

class ArticleListView(OwnerListView):
    model = Article
```

dj4e-samples/myarts/templates/myarts/article_list.html

```
<ul>
{% for article in article_list %}
<li>
<a href="{% url 'myarts:article_detail'  article.id %}">
   {{ article.title }}</a>
{% if article.owner == user %}
(<a href="{% url 'myarts:article_update' article.id %}">
   Edit</a> |
<a href="{% url 'myarts:article_delete' article.id %}">
   Delete</a>)
{% endif %}
</li>
{% endfor %}
</ul>
```



Inheritance (Review)

- When we make a new class - we can reuse an existing class and inherit all the capabilities of an existing class and then add our own little bit to make our new class
- Another form of store and reuse
- Write once - reuse many times
- The new class (child) has all the capabilities of the old class (parent) - and then some more

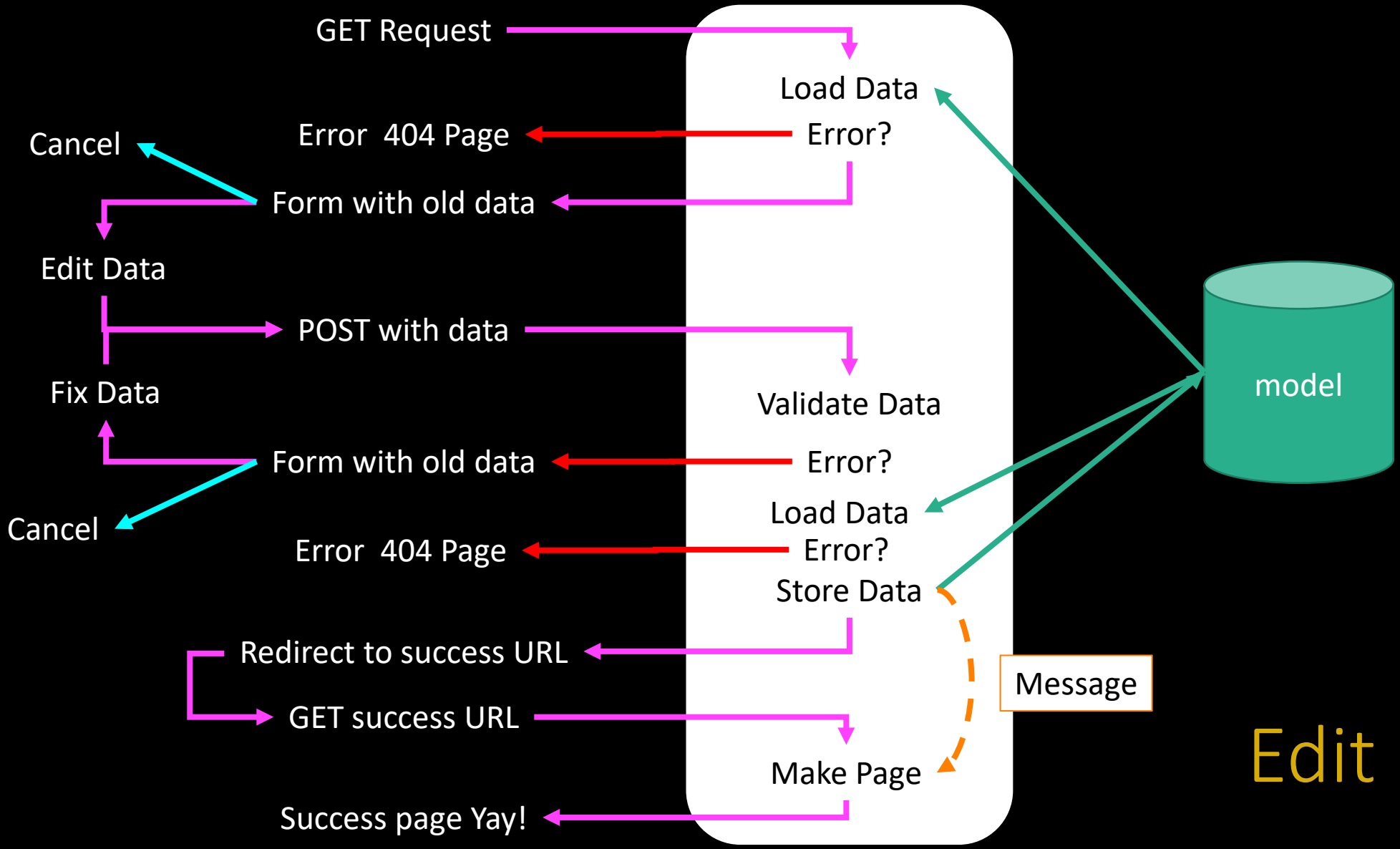
Terminology: Inheritance

‘Subclasses’ are more specialized versions of a class, which inherit attributes and behaviors from their parent classes, and can introduce their own.

http://en.wikipedia.org/wiki/Object-oriented_programming

Inside a Generic Edit View

(review)



Edit Form Flow

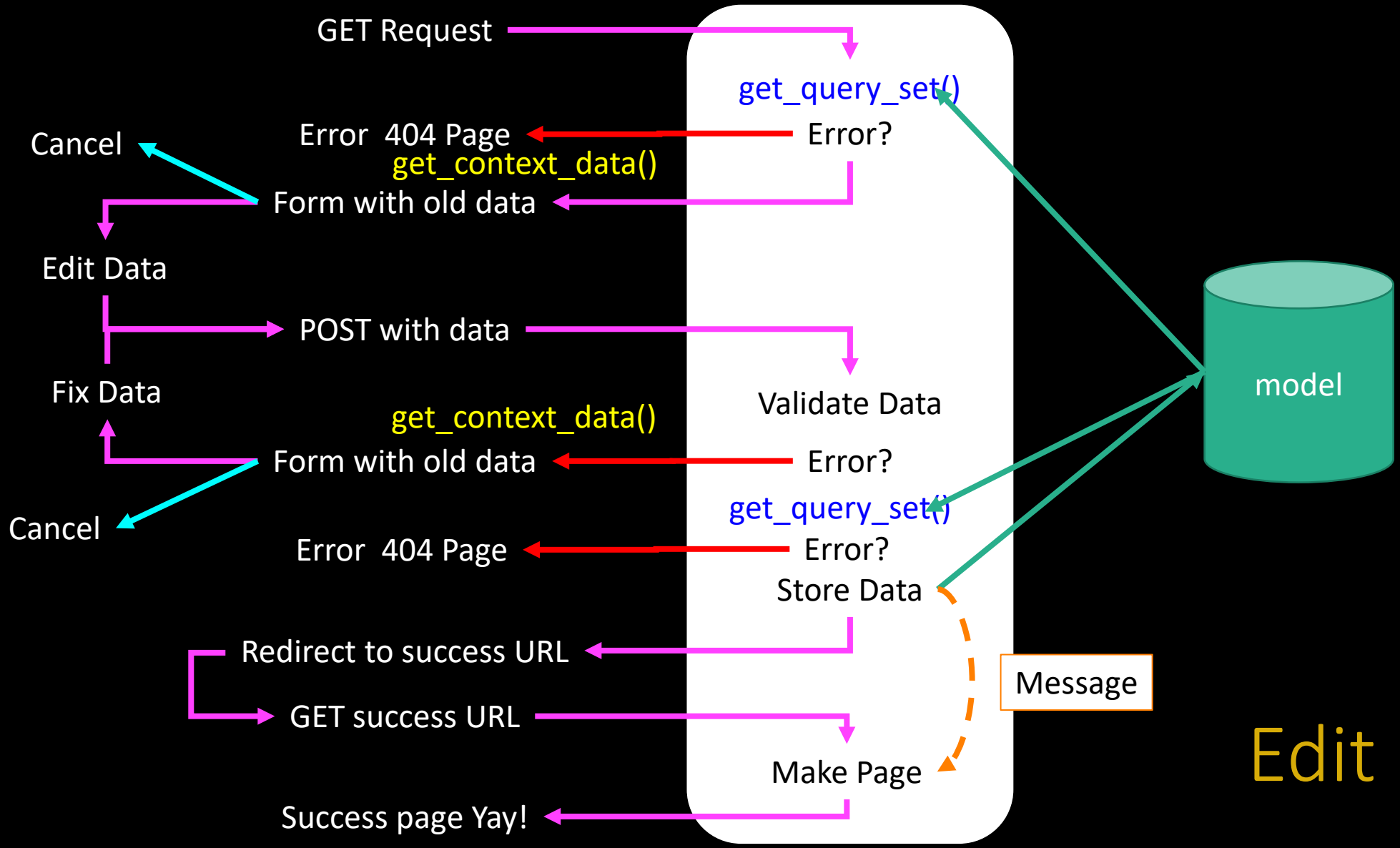
class django.views.generic.list.ListView

A page representing a list of objects. While this view is executing, self.object_list will contain the list of objects (usually, but not necessarily a queryset) that the view is operating upon.

Method Flowchart

1. setup()
2. dispatch()
3. http_method_not_allowed()
4. get_template_names()
5. get_queryset()
6. get_context_object_name()
7. get_context_data()
8. get()
9. render_to_response()

https://docs.djangoproject.com/en/3.0/ref/class-based-views/generic-display/#django.views.generic.list.ListView

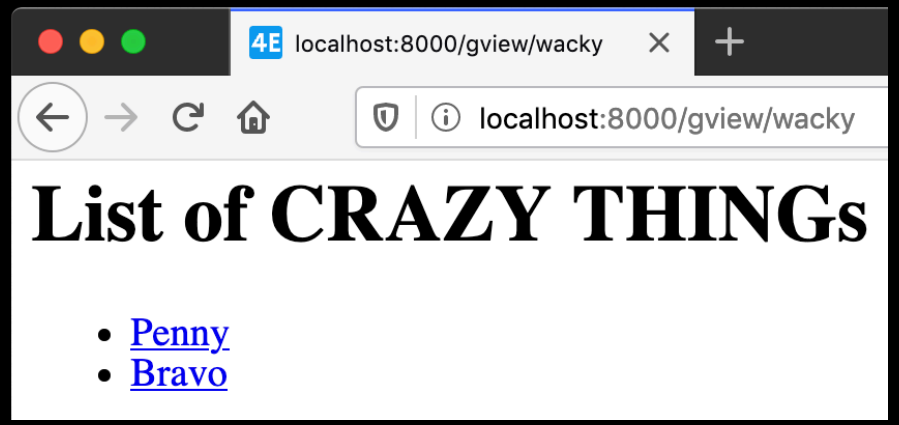


Edit Form Flow

dj4e-samples/gview/views.py

```
# Lets explore how (badly) we can override things...
class WackyEquinesView(generic.ListView):
    model = Car
    template_name = 'gview/wacky.html'

    def get_queryset(self, **kwargs):
        crazy = Horse.objects.all()     # Convention: Car
        return crazy

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['crazy_thing'] = 'CRAZY THING'
        return context
```

https://samples.dj4e.com/gview/wacky



dj4e-samples/gview/templates/gview/wacky.html

```
<h1>List of {{ crazy_thing }}s</h1>
<p>
{% if horse_list %}
<ul>
  {% for xyz in horse_list %}
    <li>
      <a href="{% url 'gview:horse' xyz.id %}">{{ xyz.name }}</a>
    </li>
  {% endfor %}
...
```

Owner List View

https://samples.dj4e.com/myarts/

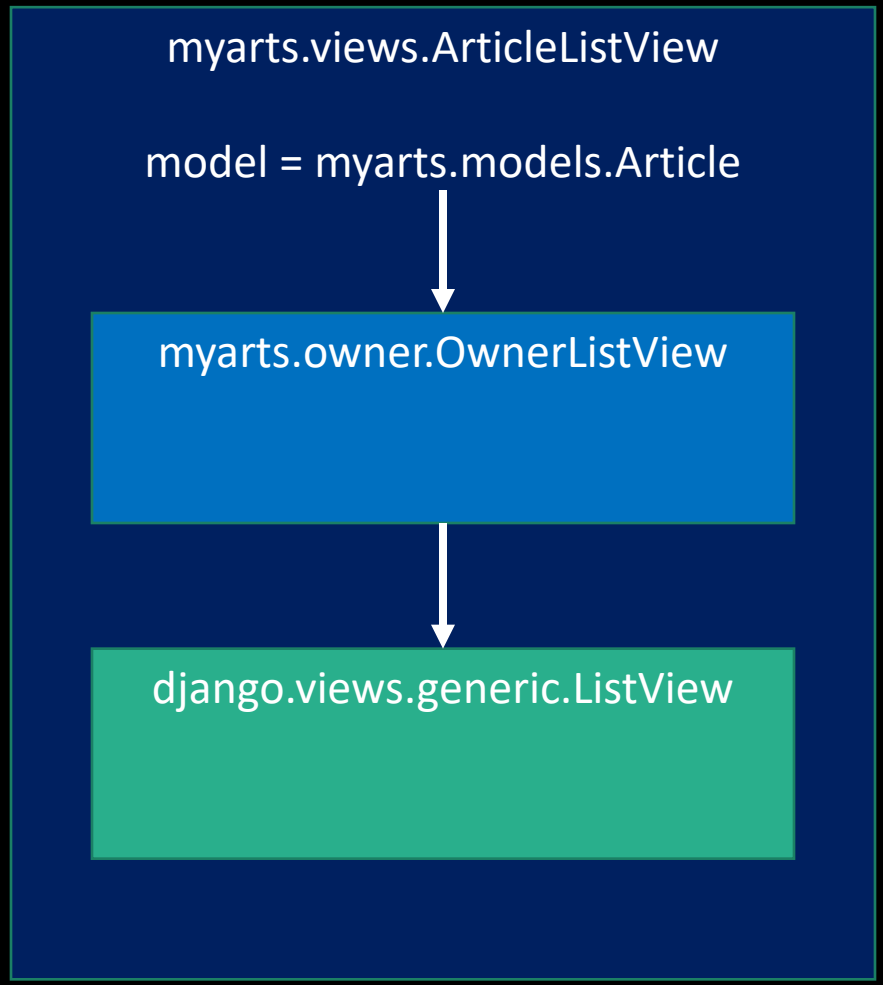
https://github.com/csev/dj4e-samples/

dj4e-samples/myarts/views.py

```
from myarts.models import Article
from myarts.owner import OwnerListView

class ArticleListView(OwnerListView):
    model = Article
```

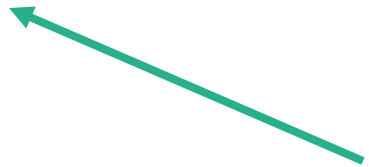
dj4e-samples/myarts/templates/myarts/article_list.html

```
<ul>
{% for article in article_list %}
<li>
<a href="{% url 'myarts:article_detail'  article.id %}">
   {{ article.title }}</a>
{% if article.owner == user %}
(<a href="{% url 'myarts:article_update' article.id %}">
   Edit</a> |
<a href="{% url 'myarts:article_delete' article.id %}">
   Delete</a>)
{% endif %}
</li>
{% endfor %}
</ul>
```



dj4e-samples/myarts/models.py

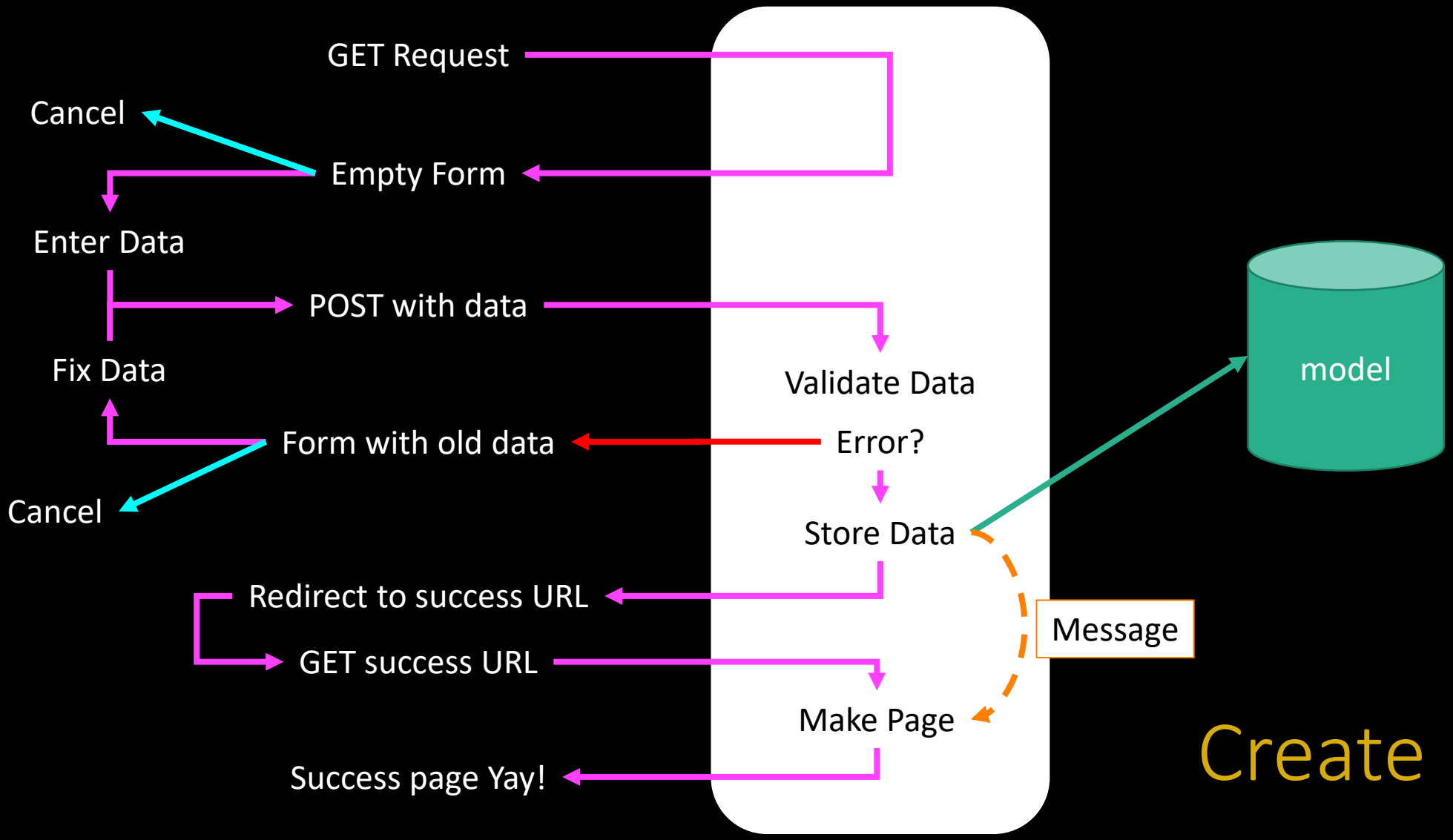
```
from django.db import models
from django.core.validators import MinLengthValidator
from django.contrib.auth.models import User
from django.conf import settings

class Article(models.Model) :
    title = models.CharField(
            max_length=200,
            validators=[MinLengthValidator(2, "Title must be greater than 2 characters")]
    )
    text = models.TextField()
    owner = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    # Shows up in the admin list
    def __str__(self):
        return self.title
```



A foreign key to a table that belongs to Django

dj4e-samples/myarts/views.py

```
from myarts.models import Article
from myarts.owner import OwnerListView,OwnerDetailView,OwnerCreateView,OwnerUpdateView,OwnerDeleteView

class ArticleListView(OwnerListView):
    model = Article
    # By convention:
    # template_name = "myarts/article_list.html"

class ArticleDetailView(OwnerDetailView):
    model = Article

class ArticleCreateView(OwnerCreateView):
    model = Article
    fields = ['title', 'text']

class ArticleUpdateView(OwnerUpdateView):
    model = Article
    fields = ['title', 'text']

class ArticleDeleteView(OwnerDeleteView):
    model = Article
```



Create Form Flow

class django.views.generic.edit.ModelFormMixin

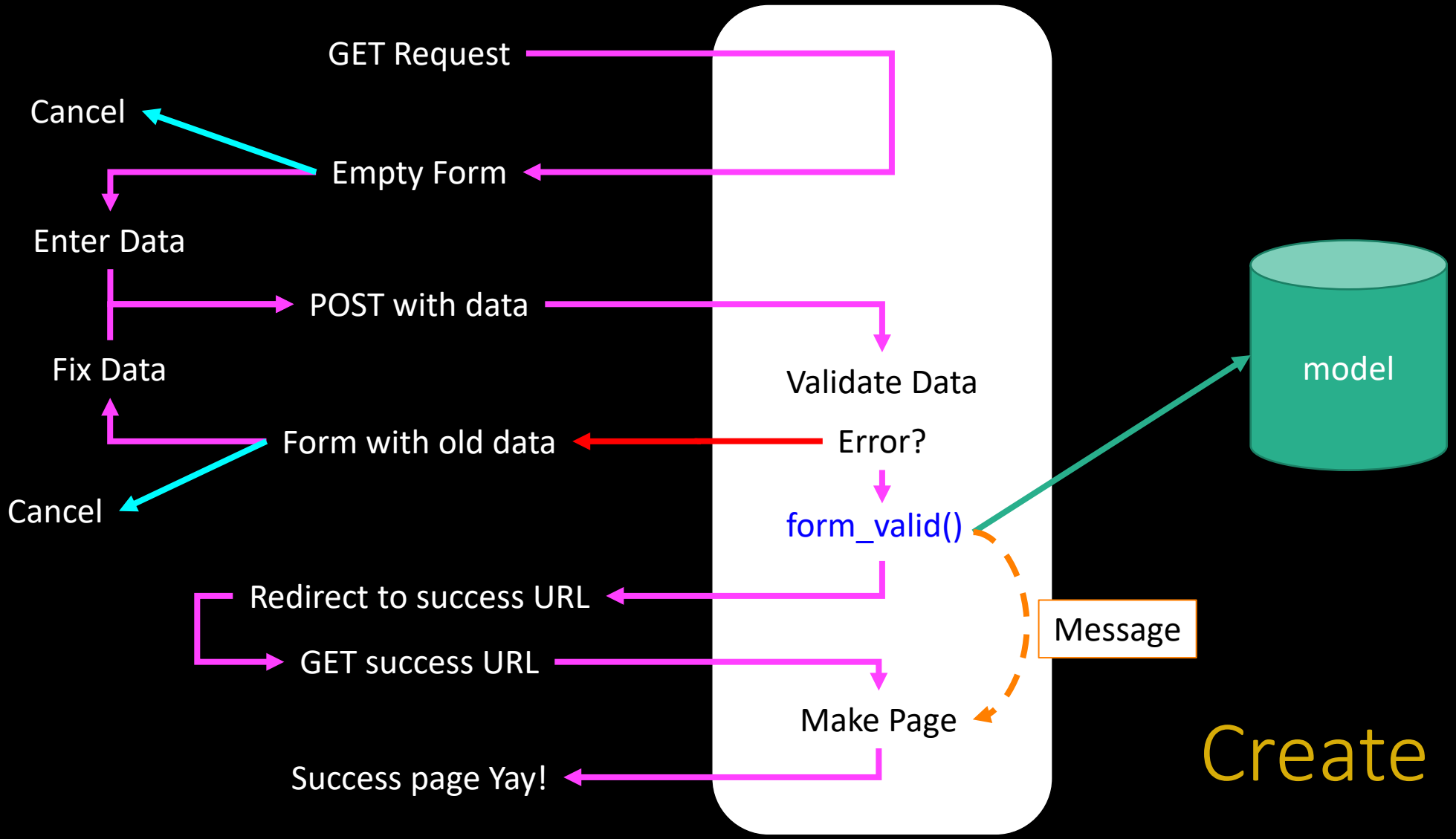
A form mixin that works on ModelForms, rather than a standalone form

get_success_url()
Determine the URL to redirect to when the form is successfully validated. Returns success_url if it is provided; otherwise, attempts to use the get_absolute_url() method of the object.

form_valid(form)
Saves the form instance, sets the current object for the view, and redirects to get_success_url().

f**orm_invalid(form)**
Renders a response, providing the invalid form as context.

https://docs.djangoproject.com/en/3.0/ref/class-based-views/mixins-editing/#modelformmixin



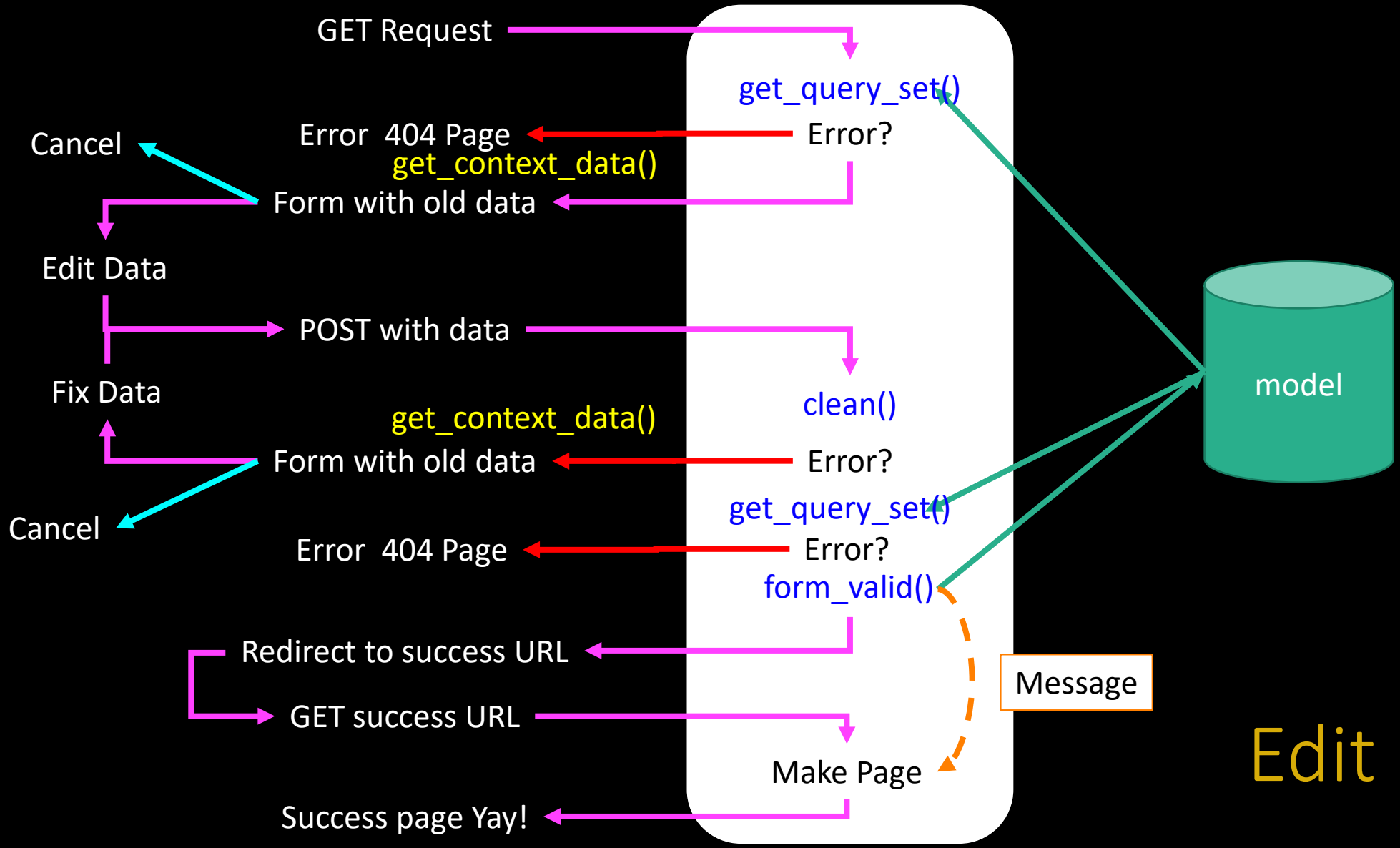
Create Form Flow

dj4e-samples/myarts/owner.py

```
from django.views.generic import CreateView, UpdateView, DeleteView, ListView, DetailView
from django.contrib.auth.mixins import LoginRequiredMixin

class OwnerCreateView(LoginRequiredMixin, CreateView):
    """
    Sub-class of the CreateView to automatically pass the Request to the Form
    and add the owner to the saved object.
    """

    # Saves the form instance, sets the current object for the
    # view, and redirects to get_success_url().
    def form_valid(self, form):
        print('form_valid called')
        object = form.save(commit=False)
        object.owner = self.request.user
        object.save()
        return super(OwnerCreateView, self).form_valid(form)
```

dj4e-samples/myarts/owner.py

```
from django.views.generic import CreateView, UpdateView, DeleteView, ListView, DetailView
from django.contrib.auth.mixins import LoginRequiredMixin

class OwnerUpdateView(LoginRequiredMixin, UpdateView):
    """
    Sub-class the UpdateView to pass the request to the form and limit the
    queryset to the requesting user.
    """

    def get_queryset(self):
        print('update get_queryset called')
        """ Limit a User to only modifying their own data. """
        qs = super(OwnerUpdateView, self).get_queryset()
        return qs.filter(owner=self.request.user)
```



Edit Form Flow

dj4e-samples/myarts/models.py

```
from django.db import models
from django.core.validators import MinLengthValidator
from django.contrib.auth.models import User
from django.conf import settings

class Article(models.Model) :
    title = ...
    text = models.TextField()
    owner = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

dj4e-samples/myarts/views.py

```
from myarts.models import Article
from myarts.owner import OwnerDeleteView

class ArticleDeleteView(OwnerDeleteView):
    # By convention, template='myarts/article_confirm_delete.html'
    model = Article
```

Summary

- We can extend the generic edit views to support an owner field in our model that is automatically populated
- By understanding and using Django in a a proper object oriented manner our code can be very simple and minimize repetition for common features
- Avoids filling views with boilerplate as the views get more complex

Acknowledgements / Contributions

Initial Development: Dr. Charles R. Severance, University of Michigan School of Information

Insert new Contributors and Translators here including names and dates

Continue new Contributors and Translators here

Additional Source Information

- Snowman Cookie Cutter" by Didriks is licensed under CC BY https://www.flickr.com/photos/dinnerseries/23570475099